Chapter 6

EXPLOITING VENDOR-DEFINED MESSAGES IN THE USB POWER DELIVERY PROTOCOL

Gunnar Alendal, Stefan Axelsson and Geir Olav Dyrkolbotn

Abstract The USB Power Delivery protocol enables USB-connected devices to negotiate power delivery and exchange data over a single connection such as a USB Type-C cable. The protocol incorporates standard commands; however, it also enables vendors to add non-standard commands called vendor-defined messages. These messages are similar to the vendor-specific commands in the SCSI protocol, which enable vendors to specify undocumented commands to implement functionality that meets their needs. Such commands can be employed to enable firmware updates, memory dumps and even backdoors.

This chapter analyzes vendor-defined message support in devices that employ the USB Power Delivery protocol, the ultimate goal being to identify messages that could be leveraged in digital forensic investigations to acquire data stored in the devices.

Keywords: USB Power Delivery protocol, vendor-specified messages, exploitation

1. Introduction

An important goal of mobile device forensics is to acquire data. Mobile phones typically have two key data sources: (i) volatile memory (RAM); and (ii) long-term storage (typically, flash memory). These two sources differ in content and acquisition methods. RAM is often proprietary, short-term storage that is not intended for interpretation by applications other than the one that stored the data. In contrast, long-term storage such as flash memory contains well-structured data, usually in a filesystem, that is meant to be re-read, typically by the operating system. Nevertheless, both types of storage maintain data that is important in digital forensic investigations.

Published by Springer Nature Switzerland AG 2019
G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics XV, IFIP AICT 569, pp. 101–118, 2019.
https://doi.org/10.1007/978-3-030-28752-8_6

Security mechanisms in commercial products are hindering the forensic acquisition of data. Data encryption in flash memory has invalidated methods such as desoldering (i.e., chip-off) that enable data to be read directly from a chip. Encryption prevents the extracted data from being interpreted without the decryption keys. The keys are often protected by additional encryption keys that tie the data to the specific device that encrypted the data in long-term storage. Therefore, transplanting a flash memory chip to a different, but identical, device would not decrypt the stored data. Device-tied encryption keys are also protected by security features such as TrustZone that rely on tamper-proof hardware. Therefore, in order to access data from a secured device, it is necessary to exploit security vulnerabilities in the device itself, or leverage undocumented features such as backdoors or indirectly increase the attack surface of the device.

The general approach is that any data extraction technique should be researched extensively, including any and all means it uses to communicate with other devices. The USB Power Delivery protocol is a communications mode that has the potential to increase the device attack surface. The idea is that, if undocumented means exist to communicate with the device, then hidden features and security vulnerabilities could be identified and exploited to facilitate data acquisition.

The USB Power Delivery protocol provides a uniform means for vendors to implement power negotiation between power sources and devices such as mobile phones and personal computers in order to maximize the charging current. The power source can support different power configurations, one power profile for a mobile phone and a different profile for a personal computer, to enable the devices to obtain the appropriate currents and voltages. Devices can also use the protocol to request higher currents and voltages from power sources. In the case of two non-power-source devices (e.g., two mobile phones), the devices can negotiate a power delivery profile so that one device can charge the other. Another example is a monitor connected to a personal computer where the protocol enables the monitor to draw power from the personal computer if it is not connected to an external power source. If the monitor is connected to an external power source, then it could provide power to the personal computer. All these negotiations occur over the same USB cable unbeknownst to the user.

The USB Power Delivery protocol is of interest from a digital forensics perspective because it supports inter-device communications. These communications could be exploited to expand the attack surface of one or both devices, enabling the acquisition of data that is otherwise inaccessible. The focus is on vendor-defined messages in the USB Power

Delivery protocol. Undocumented messages discovered in other protocols have been demonstrated to enable firmware updates, memory dumps and even backdoors. This chapter presents a black-box testing approach for revealing proprietary messages supported by the USB Power Delivery protocol that could be leveraged in digital forensic investigations to acquire data stored in devices that support the protocol.

2. Related Work

Allowing vendors to incorporate proprietary vendor-defined messages or commands in protocols to provide custom functionality has led to the release of numerous consumer devices that potentially respond to undocumented commands with unknown behavior. This can have devastating security implications. As demonstrated by Alendal et al. [2], vendor-specified SCSI commands can be used to bypass authentication on self-encrypting hard drives. Whether this research represents the best-case scenario for law enforcement or the worst-case scenario for the vendor, one cannot ignore the fact that the existence of hidden commands must be tested carefully. Indeed, as devices and firmware change over time, such testing should be performed regularly by law enforcement and security researchers.

Testing the USB Power Delivery protocol for hidden commands requires a means for emulating the protocol. Reydarns et al. [5] have demonstrated the use of USB Power Delivery protocol emulation in testing different power configurations for a power source. However, there is little, if any, research on the security of the USB Power Delivery protocol and nothing related to digital forensics. This research is important because it comprehensively analyzes the USB Power Delivery protocol and attempts to discover how vendor-defined protocol messages could be leveraged to assist digital forensic examinations of devices that support the protocol.

3. USB Power Delivery Protocol

Revision 1.0 (version 1.0) of the USB Power Delivery protocol specification was released in 2012; several revisions have been released since, the most recent being Revision 2.0 (version 1.3) and Revision 3.0 (version 1.2) [8]. The protocol provides a uniform means for devices to negotiate power supply configurations across vendors. It is typically supported by devices with a USB Type-C port/connector with dedicated CC1 and CC2 links (Figure 1). The USB Type-C connection is reversible, enabling devices to communicate on either CC line.

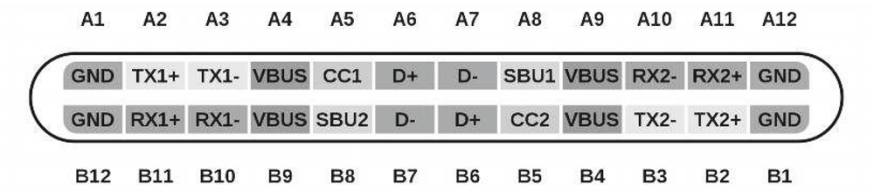


Figure 1. USB Type-C pinout [4].

The message-based USB Power Delivery protocol has three types of messages: (i) control messages; (ii) data messages; and (iii) extended messages. Control messages are short messages that typically require no data exchange. Data messages contain data objects that are transmitted between devices. Extended messages are essentially data messages with larger data payloads. The USB Power Delivery protocol leverages the three message types to define a wide range of standard messages, which enable devices to communicate and negotiate power source configurations.

Figure 2. Data message packet.

Figure 2 shows a data message packet comprising a preamble for synchronization, start of packet (SOP), message header, up to eight data objects of 32-bits each, CRC and end of packet (EOP). The preamble, SOP, CRC and EOP are part of the physical transport layer; they are common to all three types of messages, along with the message header. The optional data objects are only found in data messages.

Table 1 lists example control and data messages in the USB Power Delivery protocol.

The USB Power Delivery protocol supports different standard message sets as indicated by the protocol specification revisions, currently Revision 2.0 and Revision 3.0. Revision 3.0 is functionally the same as Revision 2.0, except for new features such as USB authentication. Interested readers are referred to the protocol specifications [8] for information pertaining to the differences between the message sets.

The USB Power Delivery protocol also enables cables to take part in communications; a device can communicate with a cable directly using the start of packet. Such electronically-marked cables (EMCA) enable devices to ensure that the cable supports high voltage/current power

Table 1. Control and data messages in Revision 3.0 (version 1.2).

Control Messages	**Data Messages**
GoodCRC	Source_Capabilities
GotoMin	Request
Accept	BIST
Reject	Sink_Capabilities
Ping	Battery_Status
PS_RDY	Alert
Get_Source_Cap	Get_Country_Info
Get_Sink_Cap	Vendor_Defined
DR_Swap	
PR_Swap	
VCONN_Swap	
Wait	
Soft_Reset	
Not_Supported	
Get_Source_Cap_Extended	
Get_Status	
FR_Swap	
Get_PPS_Status	
Get_Country_Codes	

source configurations. According to the protocol specification, devices can negotiate direct current levels up to 5 A, corresponding to a maximum of 100 W at 20 V between devices connected via an EMCA cable. Passive (non-EMCA) cables are rated for a maximum direct current of 3 A, which corresponds to 15 W at 5 V, 36 W at 12 V or 60 W at 20 V.

Figure 3 shows a typical power delivery negotiation – referred to as an explicit contract between two devices or port pairs. According to the standard, all port pairs are required to make an explicit contract. In a contract, the device (port) that consumes power is called the sink and the device (port) that provides power is called the source.

Vendors may implement novel functionality using proprietary vendor-defined messages, a subgroup of data messages in the USB Power Delivery protocol. Similar features are found in other protocols, such as vendor-specific commands in the SCSI protocol [6]. These commands are implemented and used only by vendors for internal purposes such as debugging, factory setup and proprietary communications with vendor software; the commands are not used in normal device operations. Vendor commands are rarely documented because they are reserved for internal use.

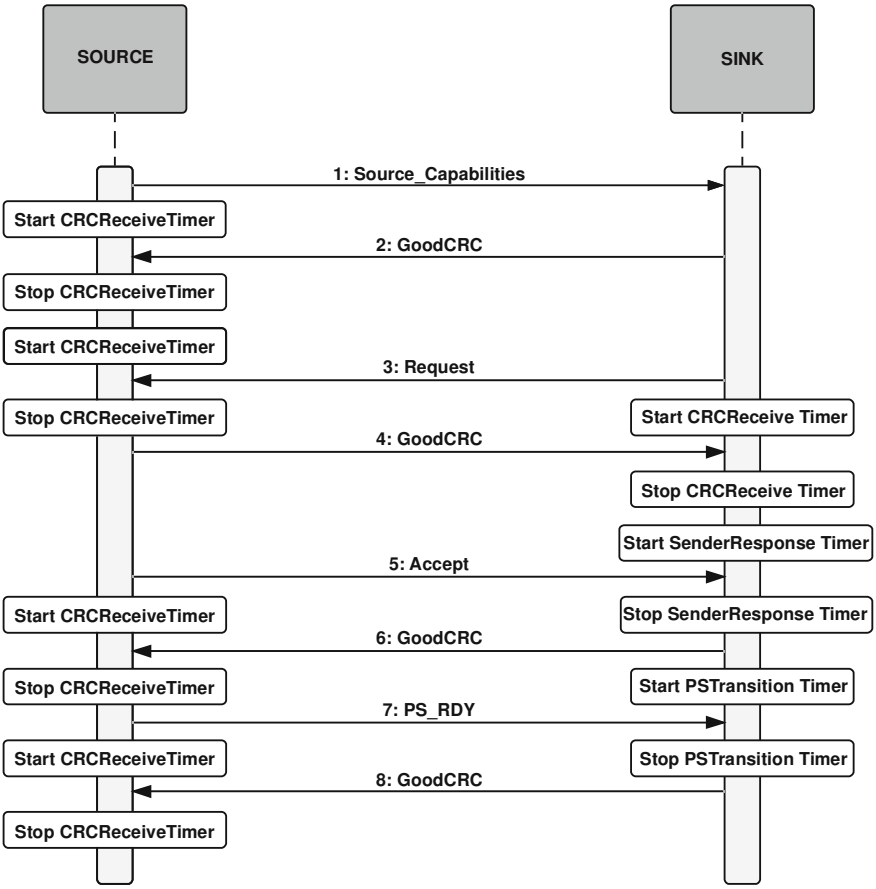


Figure 3. Simplified explicit contract negotiation.

Figure 4. Vendor-defined message packet.

Figure 4 shows a vendor-defined message (VDM) packet in the USB Power Delivery protocol. Vendor-defined messages are of two types: (i) structured; and (ii) unstructured. Structured vendor-defined message commands are defined in the USB Power Delivery protocol standard whereas unstructured vendor-defined message commands are implemented by vendors on an *ad hoc* basis. Note that a "command" is a subgroup of "message," which is either a structured vendor-defined mes-

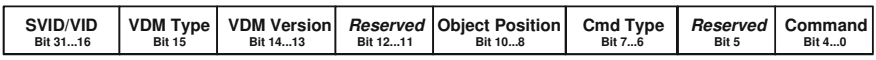


Figure 5. Structured vendor-defined message header.

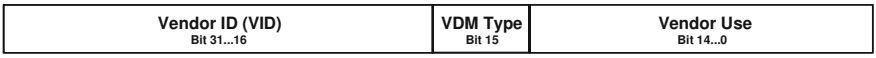


Figure 6. Unstructured vendor-defined message header.

sage or an unstructured vendor-defined message. Thus, while structured vendor-defined messages have predefined command sets in the protocol specification, unstructured vendor-defined messages can correspond to commands defined by vendors.

Because vendor-defined messages are a type of data message, there is a size limitation on the amount of data a message can contain – this corresponds to the size of six vendor data objects (VDOs) plus the 32-bit vendor-defined message header. A vendor data object contains a 32-bit value (data). To prevent vendors from implementing conflicting messages, the protocol requires either the standard vendor ID (SVID) defined in the protocol specification or a vendor ID (VID) to be part of the vendor-defined message header. This means that a vendor must use one of its 16-bit USB Implementers Forum (USB-IF) vendor IDs [7] in all the vendor-defined messages it implements.

Example vendor IDs are `0x05ac` (Apple) and `0x04e8` (Samsung). As shown in Figures 5 and 6, the structured vendor ID and vendor ID are required to be part of the corresponding vendor-defined message headers. Thus, a vendor with a valid USB-IF-assigned vendor ID can implement any command that contains up to six additional vendor data objects in one vendor-defined message. The command is the second part of the vendor-defined message header that can be any 15-bit value in the case of an unstructured vendor-defined message.

Table 2 shows example structured vendor-defined message commands.

4. Methodology

Devices come in different architectures from numerous vendors and without source code or firmware that implement the USB Power Delivery protocol. Therefore, a black-box method was attempted to test the existence of vendor-defined messages in the protocol. One approach is to analyze protocol communications between devices from the same vendor and determine if vendor-defined messages are employed. This

Table 2. Structured commands in Revision 3.0 (version 1.2).

Structured Vendor-Defined Message Commands
Discover Identity
Discover SVIDs
Discover Modes
Enter Mode
Exit Mode
Attention
SVID Specific Commands (defined by the SVID)

assumes that, if such messages exist, the connected devices initiate their use by default.

Instead, a more active approach that directly communicates with a test device was employed. Since no solution was available to communicate with devices via the USB Power Delivery protocol, a home-grown approach was employed. A detailed description of this approach is beyond the scope of this chapter. However, the concept is simple – set up a device to act as the source, establish a connection with the test device and check for vendor-defined messages.

Testing for vendor-defined messages sounds simple, but the reality is quite different. Because the protocol specification states that any vendor-defined message must include a vendor ID, it is necessary to know or guess the expected vendor ID of the device of interest. This is important because a device would not respond to a vendor-defined message containing a correctly-guessed command but an incorrect vendor ID in the header.

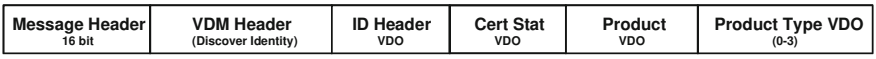


Figure 7. Discover Identity reply packet.

Fortunately, it is possible to leverage the Discover Identity command in the structured vendor-defined message command set shown in Table 2. This command is required by the USB Power Delivery protocol, so all devices should support the command. The command, which enables devices and cables to identify other end points, has a predefined reply packet format with a fixed number of vendor data objects and their content (Figure 7). The ID header of the 32-bit vendor data object has bits 0–15 reserved for the device USB-IF vendor ID. A connected device reveals its vendor ID upon receiving a Discover Identity command.

The protocol specification also states that structured vendor-defined messages shall only be used when an explicit contract is in place (except for a small number of cables that are not relevant in this context). The same holds true for unstructured vendor-defined messages. Thus, a device will not reply to a vendor-defined message until an explicit contract is in place (i.e., a power source configuration has been negotiated). Therefore, it is required to simulate a complete explicit contract negotiation with a test device before a vendor-defined message can be received.

This makes it necessary to simulate many messages (Figure 3) with corresponding time-outs, such as CRCReceiveTimer (maximum 1.1 ms), SenderResponseTimer (maximum 30 ms) and PSTransitionTimer (maxiumum 550 ms). Since the protocol defines the time-out values, the reply to a packet must be provided in time or the device will time out. Many of these requirements are strict, so the simulator must have a quick response, which, in turn, may render a pure software solution infeasible.

By negotiating an explicit contract with a device, it is possible to explore the existence of unstructured vendor-defined commands. Using the vendor ID captured from the response of a device to a Discover Identity command, different unstructured vendor-defined commands could be sent to the device and the responses, if any, could be examined. This can be done by brute forcing the lower 15 vendor use bits of the unstructured vendor-defined message header (Figure 5) with a fixed vendor ID for each device.

Two approaches are possible. The first is to attempt to measure the skews in the timing of device responses. The second is to test for device responses other than the expected GoodCRC message. Testing for timing skews could indicate that the device spent additional time to process a correctly-guessed unstructured vendor-defined command. However, this approach requires high resolution timers. Unfortunately, the experimental setup could only measure the time elapsed from when a packet was sent to when the response was received, which was much too inaccurate. Therefore, the second approach involving device responses other than the expected GoodCRC message was employed in the experiments.

5. Experimental Results

Not every device with a USB Type-C connector is enabled for the USB Power Delivery protocol. If a test device with a USB Type-C connector does not respond with a GoodCRC message to the Source_Capabilities

Table 3. Test devices with USB Type-C connectors and protocol support.

Device (Model)	Firmware Version	Protocol Revision	Exposed Vendor ID
HTC 10 (2PS6200)	1.90.401.5	2.0	`0x0bb4` (HTC)
HTC U11 (2PZC100)	1.13.401.1	3.0	`0x05c4` (Qualcomm)
Huawei Mate 10 Pro (BLA-L29)	8.0.0.137(C432)	2.0	`0x12d1` (Huawei)
LG G5 (LG-H850)	V10i-EUR-XX MMB29M	2.0	`0x0000` (Unknown)
Nokia 8 Sirocco (TA-1005)	00WW_3_10F	2.0	`0x05c6` (Qualcomm)
Samsung Galaxy S9 (G960F)	G960FXXU2BRH7	3.0	`0x04e8` (Samsung)

message in an explicit contract negotiation (Figure 3), then the device can be assumed to be non-protocol-enabled.

According to Section 6.2.1.1.5 of USB Power Delivery Protocol Specification Revision 3.0 (v.1.2) [8], the source shall set its highest supported specification revision in the specification revision field of the Source_Capabilities message and the sink shall reply with its highest supported specification revision in the specification revision field of the Request message (Figure 3). Because the specification states that the specification revision field value should be backwards compatible, this means the highest version can always be simulated in the first Source_Capabilities message acting as the source and the Request response from the device can then be checked.

After negotiating a complete explicit contract (Figure 3) with a test device, a Discover Identity message was sent to the device to obtain the USB-IF vendor ID from the device. Table 3 shows the test devices with USB Type-C connectors that were determined via this technique to support the USB Power Delivery protocol.

With an explicit contract in place with a test device with protocol support and its USB-IF vendor ID known, the next step was to send arbitrary protocol messages to the device and test the responses. Specifically, unstructured vendor-defined messages were sent with the vendor ID set to the appropriate value, type set to `0` (i.e., unstructured) and vendor use set to different values corresponding to commands (Figure 5).

Table 4. Huawei Mate 10 Pro (BLA-L29) message capture.

ID	Time	Role	Message	Data
284	0:41.044.922		Hard Reset	
286	0:43.577.218	Source:DFP	[0]Source_Cap	A1 11 F0 90 01 08 FE CA B7 52
290	0:43.577.879	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
293	0:43.580.754	Sink:UFP	[0]Request	42 10 C8 20 03 13 52 0F 95 B7
297	0:43.581.374	Source:DFP	[0]GoodCRC	A1 01 C1 AF C2 81
300	0:43.582.060	Source:DFP	[1]Accept	63 03 21 7B 00 96
303	0:43.582.586	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
306	0:43.583.283	Source:DFP	[2]PS_RDY	A6 05 1F FD EE C9
309	0:43.583.915	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
312	0:43.737.641	Source:DFP	[0]VDM:DiscIdentity	6F 11 01 80 00 FF 76 31 6B 61
316	0:43.738.185	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
319	0:43.744.295	Sink:UFP	[1]VDM:DiscIdentity	4F 52 41 80 00 FF D1 12 00 EC 00 00 00 00 00 00 7E 10 01 00 00 11 80 C1 C7 56
327	0:43.745.502	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
330	0:44.918.448	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 D1 12 0D 13 06 BC
334	0:44.919.214	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
337	0:46.507.375	Source:DFP	[2]VDM:Unstructured	6F 15 02 00 D1 12 43 49 F3 21
341	0:46.507.960	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF

The responses were analyzed and any response other than the expected GoodCRC was assumed to be an attempt by the test device to reply to the random "command" it received.

A commercial USB Power Delivery protocol recorder was used to capture communications with the test devices. Table 4 shows an example capture of messages to and from the Huawei test device that was configured as the sink. The message capture shows the entire explicit contract negotiation (message IDs 286–309) and the USB-IF vendor ID discovery (message IDs 312–327), which are followed by two unstructured vendor-defined message brute force attempts (message IDs 330–334 and message IDs 337–341). Note that the Huawei device did not respond to the two unstructured vendor-defined message tests with anything other than the expected GoodCRC message.

Very few test devices responded to the brute force test. In fact, only the Samsung device replied with anything other than a GoodCRC message, and only for some messages.

Table 5 shows an example capture of messages to and from the Samsung Galaxy S9 test device that was configured as the sink. Once again, the message capture shows the entire explicit contract negotiation (message IDs 5442–5465) and the USB-IF vendor ID discovery (message IDs

Table 5. Samsung Galaxy S9 (G960F) message capture.

ID	Time	Role	Message	Data
5440	14:36.248.230		Hard Reset	
5442	14:39.309.886	Source:DFP	[0]Source_Cap	A1 11 F0 90 01 08 FE CA B7 52
5446	14:39.310.395	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
5449	14:39.311.982	Sink:UFP	[0]Request	82 10 F0 C0 03 13 08 11 00 3A
5453	14:39.312.708	Source:DFP	[0]GoodCRC	A1 01 C1 AF C2 81
5456	14:39.313.284	Source:DFP	[1]Accept	63 03 21 7B 00 96
5459	14:39.313.979	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
5462	14:39.314.462	Source:DFP	[2]PS_RDY	A6 05 1F FD EE C9
5465	14:39.315.049	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
5468	14:39.471.248	Source:DFP	[0]VDM:DiscIdentity	6F 11 01 80 00 FF 76 31 6B 61
5472	14:39.471.866	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
5475	14:39.476.288	Sink:UFP	[1]VDM:DiscIdentity	8F 42 41 80 00 FF E8 04 00 D1 00 00 00 00 00 00 60 68 C2 B2 A2 9E
5482	14:39.477.131	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
5485	14:40.650.372	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 E8 04 E6 2B 56 46
5489	14:40.651.199	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
5492	14:40.654.796	Sink:UFP	[2]VDM:Unstructured	4F 14 41 00 E8 04 FD AA CE 68
5496	14:40.655.473	Source:DFP	[2]GoodCRC	61 05 96 BC 55 4D
5499	14:41.828.228	Source:DFP	[2]VDM:Unstructured	6F 15 02 00 E8 04 A8 71 A3 DB
5503	14:41.829.056	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
5506	14:41.833.325	Sink:UFP	[3]VDM:Unstructured	4F 56 42 00 E8 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 34 A1 0A 25
5514	14:41.834.581	Source:DFP	[3]GoodCRC	61 07 BA DD 5B A3
5517	14:43.008.455	Source:DFP	[3]VDM:Unstructured	6F 17 02 00 E8 04 C8 22 63 A1
5521	14:43.009.071	Sink:UFP	[3]GoodCRC	41 06 8E C9 D8 41
5524	14:43.013.435	Sink:UFP	[4]VDM:Unstructured	4F 58 42 00 E8 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 84 AD C5 F6
5532	14:43.014.693	Source:DFP	[4]GoodCRC	61 09 BD F0 E3 44
5535	14:44.180.619	Source:DFP	[4]VDM:Unstructured	6F 19 03 00 E8 04 CC FB EF A6
5539	14:44.181.134	Sink:UFP	[4]GoodCRC	41 08 89 E4 60 A6
5542	14:45.761.683	Source:DFP	[5]VDM:Unstructured	6F 1B 02 00 E8 04 C9 CF 93 64
5546	14:45.762.289	Sink:UFP	[5]GoodCRC	41 0A A5 85 6E 48
5549	14:45.766.649	Sink:UFP	[5]VDM:Unstructured	4F 5A 42 00 E8 04 0D DA 95 63 4A 97 17 B5 F5 34 11 47 53 7E C9 E9 8C 35 3F 0E
5557	14:45.767.917	Source:DFP	[5]GoodCRC	61 0B 91 91 ED AA
5560	14:46.933.424	Source:DFP	[6]VDM:Unstructured	6F 1D 01 00 E8 04 87 95 66 F9
5564	14:46.934.042	Sink:UFP	[6]GoodCRC	41 0C 90 20 0D A1
5567	14:46.937.851	Sink:UFP	[6]VDM:Unstructured	4F 1C 41 00 E8 04 3C E1 BE 58
5571	14:46.938.566	Source:DFP	[6]GoodCRC	61 0D A4 34 8E 43
5574	14:48.114.825	Source:DFP	[7]VDM:Unstructured	6F 1F 02 00 E8 04 09 69 13 91
5578	14:48.115.442	Sink:UFP	[7]GoodCRC	41 0E BC 41 03 4F
5581	14:48.119.820	Sink:UFP	[7]VDM:Unstructured	4F 5E 42 00 E8 04 0D DA 95 63 4A 97 17 B5 F5 34 11 47 53 7E C9 E9 37 31 C6 1C
5589	14:48.121.075	Source:DFP	[7]GoodCRC	61 0F 88 55 80 AD
5592	14:49.303.445	Source:DFP	[0]VDM:Unstructured	6F 11 03 00 E8 04 0D B0 9F 96
5596	14:49.304.274	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
5599	14:50.881.168	Source:DFP	[1]VDM:Unstructured	6F 13 02 00 E8 04 08 84 E3 54
5603	14:50.881.789	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
5606	14:50.886.156	Sink:UFP	[0]VDM:Unstructured	4F 50 42 00 E8 04 60 B3 A9 5A 65 3F 48 3C 3A D6 13 DC 2D 32 8D 16 F6 75 A3 FE
5614	14:50.887.366	Source:DFP	[0]GoodCRC	61 01 8F 78 38 4A

5468–5482). These are followed by the first unstructured vendor-defined message test (message ID 5485). The sent message has an unstructured vendor-defined message header of `0x04e80001`, which is decoded according to Figure 5 as vendor ID: `0x04e8`, type: `0` and vendor use: `0x0001` (15-bit value).

Note that this unstructured vendor-defined message received a response other that the GoodCRC (message ID 5492). The response has an unstructured vendor-defined message header of `0x04e80041`, which is decoded according to Figure 5 as vendor ID: `0x04e8`, type: `0` and vendor use: `0x0041`. This message appears to be a reply with no additional data (i.e., vendor data objects).

A similar situation is seen for message 5499 with vendor use: `0x0002`, whose response (message ID 5506) has vendor use: `0x0042` and four additional vendor data objects: `0x00000000 0x00000000 0x00000000` and `0x00000000`.

The two vendor use command/reply pairs of `0x0001`/`0x0041` and `0x0002`/`0x0042` imply that bit 6 (`0x0040`) may be an ACK bit. If the unstructured headers are interpreted as structured headers (Figure 6), then bits 6–7 correspond to type where `0x1` (bit 6 set) corresponds to an ACK. Of course, the real situation is not clear, but it does appear that the vendor may have mixed the two types of vendor-defined message headers.

Investigating further, the response (message ID 5506) with vendor use set to `0x0042` also has four additional vendor data objects: `0x00000000 0x00000000 0x00000000` and `0x00000000`. This appears to be data sent back to the source side from the sink. All the vendor data objects contain zeroes in the replies to two consecutive messages with vendor use set to `0x0002` (message IDs 5499 and 5517).

However, when a different message (message ID 5535) is sent to the device with vendor use set to `0x0003`, then a completely different reply is received with vendor use set to `0x0002` (message ID 5542) and four vendor data objects: `0x6395da0d 0xb517974a 0x471134f5` and `0xe9c97e53` (message ID 5549). Sending message 5535 again (message ID 5574) yields the same four vendor data objects (message ID 5581). However, another message with vendor use set to `0x0003` (message ID 5592) once again changes the vendor data objects for vendor use set to `0x0002`. Specifically, the four vendor data objects are: `0x5aa9b360 0x3c483f65 0xdc13d63a` and `0x168d322d` (message ID 5606).

It appears that data in the form of vendor data objects is received from the device and different data is received when sending a specific message with vendor use set to `0x0003`. The four vendor data objects appear to change in pseudorandom order. Another observation is that,

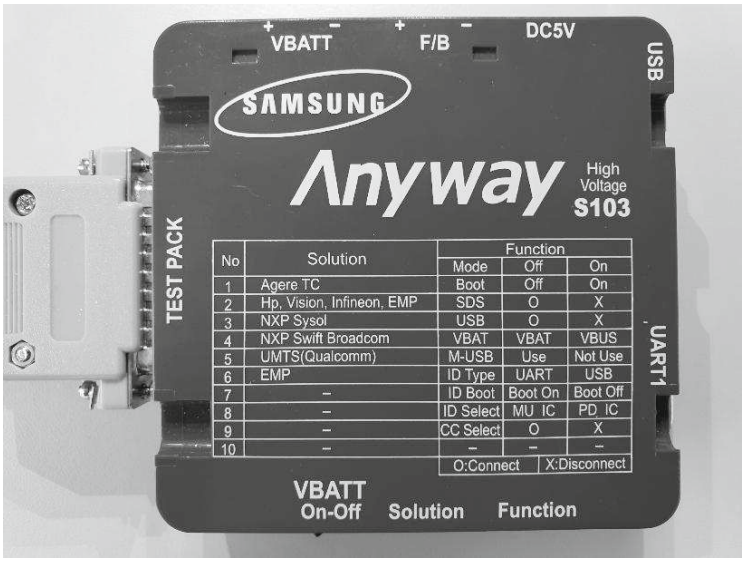
Table 6. Samsung Galaxy S9 (G960F) message capture.

ID	Time	Role	Message	Data
162	0:06.589.154	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 E8 04 E6 2B 56 46
166	0:06.589.982	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
169	0:06.594.059	Sink:UFP	[1]VDM:Unstructured	4F 12 41 00 E8 04 5D 5F 8E E7
173	0:06.594.675	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
176	0:06.629.222	Source:DFP	[2]VDM:Unstructured	6F 55 02 00 E8 04 1C 47 B3 AB 2E F3 7B AE
				F9 09 79 82 02 3B C6 BB 1A D4 E8 41
184	0:06.630.376	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
187	0:06.635.264	Sink:UFP	[2]VDM:Unstructured	4F 54 42 00 E8 04 1C 47 B3 AB 2E F3 7B AE
				F9 09 79 82 02 3B C6 BB 51 65 55 63
195	0:06.636.524	Source:DFP	[2]GoodCRC	61 05 96 BC 55 4D

when a message is sent with vendor use set to `0x0002` along with four random vendor data objects (`0xabb3471c`, `0xae7bf32e`, `0x827909f9`, `0xbbc63b02`), a reply is received with the same vendor data objects (Table 6). This implies that a message with vendor use set to `0x0002` corresponds to an initialization command. Repeating the messages with vendor use set to `0x0003` and `0x0002` gives different vendor data objects, which may correspond to some form of encryption or obfuscation.

Sending two identical runs of the messages in Table 5 gives the same results and any randomization of the four vendor data objects sent with vendor use set to `0x0002` yields seemingly random reply vendor data objects when intermingled with messages with vendor use set to `0x0003`. This strengthens the belief that encryption is in place and that the message with vendor use set to `0x0002` is either transmitting a key or an initialization vector for a symmetric cipher.

Because the results indicate that Samsung devices respond to vendor-defined messages in the USB Power Delivery protocol, additional experiments were conducted to confirm the results. The experiments employed a special factory test device called the Samsung Anyway S103 (Figure 8). This device enables a console interface provided by the device bootloader, which is useful for debug logging and other activities. The same console can be reached via a custom USB connector and a simple RS232-to-USB serial converter on older devices with micro-USB connectors [3]. Alendal et al. [1] employed this type of connection to demonstrate an exploit targeting Samsung devices with a certain security vulnerability. The exploit assisted in bypassing a security feature in the devices. This demonstrates the importance of expanding the attack surface of a device by enabling the factory test feature.



No	Solution	Function		
		Mode	Off	On
1	Agere TC	Boot	Off	On
2	Hp, Vision, Infineon, EMP	SDS	O	X
3	NXP Sysol	USB	O	X
4	NXP Swift Broadcom	VBAT	VBAT	VBUS
5	UMTS(Qualcomm)	M-USB	Use	Not Use
6	EMP	ID Type	UART	USB
7	–	ID Boot	Boot On	Boot Off
8	–	ID Select	MU IC	PD IC
9	–	CC Select	O	X
10	–	–	–	–
		O:Connect		X:Disconnect

Figure 8. Samsung Anyway S103.

The special factory device was hard to obtain because it is usually provided to Samsung device repair shops and similar outlets. However, a factory device was procured to communicate with the Samsung test device using the USB Power Delivery protocol. Table 7 shows a message capture with the Samsung Anyway S103 and Samsung Galaxy S9 configured as the source and sink, respectively (the vendor data objects are partially redacted). Note that the communications in the message capture did not involve an explicit contract negotiation as required in the protocol specification. Instead, immediate vendor-defined message communications were conducted using the discovered vendor-defined messages. The capture corresponds to a vendor-defined message with vendor use set to `0x0001`, followed by a vendor-defined message with vendor use set to `0x0002` that provides four pseudorandom vendor data objects. These are followed by several vendor-defined messages with vendor use set to `0x0003`, each containing four vendor data objects with seemingly pseudorandom data.

Next, the Samsung Anyway S103 factory device was removed as the source and a blind replay from the source side of the communications was attempted. The idea was that, if the source messages from the Samsung Anyway S103 device were replayed and the same sink messages were received from the test device, then the Samsung Anyway S103 device was essentially being emulated. This test was an immediate success.

Table 7. Samsung Anyway S103 and Samsung Galaxy S9 message capture.

ID	Time	Role	Message	Data
1	0:03.900.730	Source:DFP	[0]VDM:DiscIdentity	6F 11 01 80 00 FF 76 31 6B 61
5	0:03.901.546	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
8	0:03.905.272	Sink:UFP	[0]VDM:DiscIdentity	8F 40 41 80 00 FF E8 04 00 D1 00 00 00 00 00 00 60 68 05 22 9E 4A
15	0:03.906.336	Source:DFP	[0]GoodCRC	61 01 8F 78 38 4A
18	0:03.906.881	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 E8 04 E6 2B 56 46
22	0:03.907.590	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
25	0:03.912.440	Sink:UFP	[1]VDM:Unstructured	4F 12 41 00 E8 04 5D 5F 8E E7
29	0:03.913.109	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
32	0:03.913.649	Source:DFP	[2]VDM:Unstructured	6F 55 02 00 E8 04 0C DD BB FF REDACTED
40	0:03.914.888	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
43	0:03.919.998	Sink:UFP	[2]VDM:Unstructured	4F 54 42 00 E8 04 0C DD BB FF REDACTED
51	0:03.921.093	Source:DFP	[2]GoodCRC	61 05 96 BC 55 4D
54	0:03.922.149	Source:DFP	[3]VDM:Unstructured	6F 57 03 00 E8 04 E6 A9 7F 72 94 CE B1
				B6 54 BA B7 75 6A F1 89 B8 01 65 20 E8
62	0:03.923.388	Sink:UFP	[3]GoodCRC	41 06 8E C9 D8 41
65	0:03.931.556	Sink:UFP	[3]VDM:Unstructured	4F 56 43 00 E8 04 9F B2 F5 F9 F1 68 E2 AF E5 AA 22 73 D0 77 6A 2E B6 3A A9 FB
73	0:03.932.759	Source:DFP	[3]GoodCRC	61 07 BA DD 5B A3
76	0:03.934.596	Source:DFP	[4]VDM:Unstructured	6F 59 03 00 E8 04 F7 96 A6 2A 08 BB A9 6E 38 40 E4 AF 33 43 7A 23 E6 D7 A8 E9
84	0:03.935.837	Sink:UFP	[4]GoodCRC	41 08 89 E4 60 A6
87	0:03.942.701	Sink:UFP	[4]VDM:Unstructured	4F 58 43 00 E8 04 9A 01 DB AE 9A 39 26 77 B0 A8 2D 11 A2 C1 76 80 1E 08 1E C2
95	0:03.943.902	Source:DFP	[4]GoodCRC	61 09 BD F0 E3 44

The key result is that the same console reached on micro-USB Samsung devices was enabled without the assistance of the Samsung Anyway S103 factory device.

The successful message replay strengthens the belief that encryption is involved and that the first four vendor data objects in the vendor-defined message with vendor use set to `0x0002` are crucial to initialization. These vendor data objects could correspond to an initialization vector or perhaps even the key to a symmetric cipher. However, experiments with several symmetric ciphers using the four vendor data objects as the key to decrypt vendor data objects in messages with the vendor use set to `0x0003` did not yield positive results.

6. Conclusions

The principal contribution of this research is a testing methodology and implementation for revealing and analyzing proprietary USB Power Delivery protocol messages. The experimental results demonstrate that at least one common mobile device, the Samsung Galaxy S9, is amenable to the testing methodology. In particular, the device responds to certain vendor-defined messages and the responses indicate the use of encryption, which raises the possibility of capturing initialization vectors and keys for symmetric ciphers. Another important result is the ability to enable factory device features in a test device in order to obtain valuable log data from the device and to widen its attack surface.

Future research will continue the investigation of vendor-defined messages in the USB Power Delivery protocol. Since vendors may also implement hidden features in other parts of the protocol, a promising approach is to investigate the role of the sink device that consumes power. Connecting two devices that typically serve as sinks – like two mobile phones – causes one device to assume the source role and provide power to the other device. This source-sink relationship could be exploited to expand the attack surface or even to directly acquire data.

Future research will also investigate potential security vulnerabilities. This is challenging because it is not known how to instrument a USB Power Delivery chip for feedback (e.g., if it crashes or demonstrates anomalous behavior). An alternative approach is to conduct a source code review or extract the chip firmware and apply reverse engineering techniques. Another approach is to analyze device-side communications with the USB Power Delivery chip, which could reveal interesting features or vulnerabilities in the chip logic as well in the operating system.

The popularity of USB Type-C connectors is increasing and large numbers of consumer devices will support the USB Power Delivery protocol. It is hoped that this work will stimulate research on the protocol and its implementations to advance device security and forensics.

Acknowledgement

This research was sponsored by the Norwegian Research Council IKTPLUSS Program under the Ars Forensica Project No. 248094/O70.

References

[1] G. Alendal, G. Dyrkolbotn and S. Axelsson, Forensic acquisition – Analysis and circumvention of Samsung secure boot enforced common criteria mode, *Digital Investigation*, vol. 24(S), pp. S60–S67,

2018.

[2] G. Alendal, C. Kison and modg, Got HW Crypto? On the (In)Security of a Self-Encrypting Drive Series, Cryptology ePrint Archive, Report 2015/1002 (eprint.iacr.org/2015/1002), 2015.

[3] N. Artenstein, Exploiting Android S-Boot: Getting arbitrary code exec in the Samsung bootloader (1/2), *Information Security Newspaper*, March 3, 2017.

[4] Chindi.ap (commons.wikimedia.org/wiki/User:Chindi.ap), 2019.

[5] H. Reydarns, V. Lauwereys, D. Haeseldonckx, P. van Willigenburg, J. Woudstra and S. De Jonge, The development of a proof of concept for a smart DC/DC power plug based on USB Power Delivery, *Proceedings of the Twenty-Second Conference on the Domestic Use of Energy*, 2014.

[6] T10 Technical Committee of the International Committee on Information Technology Standards, SCSI Operation Codes (www.t10.org/lists/op-num.htm), 2015.

[7] USB Implementers Forum, Getting a Vendor ID, Beaverton, Oregon (www.usb.org/getting-vendor-id), 2019.

[8] USB Implementers Forum, USB Power Delivery, Beaverton, Oregon (www.usb.org/document-library/usb-power-delivery), 2019.